Contents & Goals

Last Lecture:
- Basic causality model
- Ether/event pool
- System configuration

This Lecture:
- Educational Objectives: Capabilities for following tasks/questions.
  - What does this State Machine mean? What happens if I inject this event?
  - Can you please model the following behaviour.
  - What is: Signal, Event, Ether, Transformer, Step, RTC.

- Content:
  - System configuration cont’d
  - Transformers
  - Step, Run-to-Completion Step
Definition. Let $\mathcal{S}_0 = (T_0, C_0, V_0, atr_0, Eth)$ be a signature with signals, $\mathcal{D}_0$ a structure of $\mathcal{S}_0$, $(Eth, ready, \oplus, \ominus, [\cdot])$ an ether over $\mathcal{S}_0$ and $\mathcal{D}_0$. Furthermore assume there is one core state machine $M_C$ per class $C \in \mathcal{C}$.

A \emph{system configuration} over $\mathcal{S}_0$, $\mathcal{D}_0$, and $Eth$ is a pair

$$(\sigma, \varepsilon) \in \Sigma_{\mathcal{D}_0} \times Eth$$

where

- $\mathcal{S} = (\mathcal{S}_0 \uplus \{SM_C \mid C \in \mathcal{C}_0\}, \mathcal{C}_0)$,
- $V_0 \cup \{\text{stable} : \text{Bool}, -, \text{true}, \emptyset\}$
- $\cup \{ \text{stC} : SM_C, +, s_0, \emptyset \mid C \in \mathcal{C}\}$
- $\cup \{ \text{params}_E : E_{0,1}, +, 0, \emptyset \mid E \in \mathcal{E}_0\}$
- $\{C \mapsto atr_0(C)\}$
- $\cup \{\text{stable}, \text{stC}\} \cup \{\text{params}_E : E \in \mathcal{E}_0\}$
- $\{\mathcal{C} \in \mathcal{C}_0\}$

- $\mathcal{D} = \mathcal{D}_0 \uplus \{SM_C \mapsto S(M_C) \mid C \in \mathcal{C}\}$, and
- $\sigma(u)(r) \cap \mathcal{D}(\mathcal{E}_0) = \emptyset$ for each $u \in \text{dom}(\sigma)$ and $r \in V_0$. 
System Configuration: Example

\[ S_0 = (T_0, C_0, V_0, atr_0, E), \quad \mathcal{S}_0: \quad (\sigma, \epsilon) \in \mathcal{S}_0 \times \mathcal{E} \]

A system configuration is a pair \((\sigma, \epsilon)\) which comprises a system state \(\sigma\) wrt. \(\mathcal{S}\) (not wrt. \(\mathcal{S}_0\)).

Such a system state \(\sigma\) provides, for each object \(u \in \text{dom}(\sigma)\),

- values for the explicit attributes in \(V_0\),
- values for a number of implicit attributes, namely
  - a stability flag, i.e. \(\sigma(u)(\text{stable})\) is a boolean value,
  - a current (state machine) state, i.e. \(\sigma(u)(\text{st})\) denotes one of the states of core state machine \(M_C\),
  - a temporary association to access event parameters for each class, i.e. \(\sigma(u)(\text{params}_E)\) is defined for each \(E \in \mathcal{E}\).

For convenience require: there is no link to an event except for \(\text{params}_E\).
Stability

Definition.
Let $(\sigma, \varepsilon)$ be a system configuration over some $\mathcal{S}_0, \mathcal{D}_0, \mathcal{E}_0$.
We call an object $u \in \text{dom}(\sigma) \cap \mathcal{D}(\mathcal{C}_0)$ stable in $\sigma$ if and only if
$$\sigma(u)(\text{stable}) = \text{true}.$$
Recall

- The (simplified) syntax of transition annotations:

\[
\text{annot ::= [ (event) [ '!' (guard) '!' ] [ '/' (action) ] ]}
\]

- **Clear**: (event) is from \( \mathcal{E} \) of the corresponding signature.
- **But**: What are (guard) and (action)?

UML can be viewed as being parameterized in *expression language* (providing (guard)) and *action language* (providing (action)).

- **Examples**:
  - **Expression Language**:
    - OCL
    - Java, C++, ...expressions
    - ...
  - **Action Language**:
    - UML Action Semantics, "Executable UML"
    - Java, C++, ...statements (plus some event send action)
    - ...
    - ...
**Needed: Semantics**

In the following, we assume that we’re **given**

- an **expression language** $\text{Expr}$ for guards, and
- an **action language** $\text{Act}$ for actions,

and that we’re **given**

- a **semantics** for boolean expressions in form of a partial function

\[ I[\cdot](\cdot, \cdot) : \text{Expr} \times \Sigma^D_S \times D \mapsto \mathbb{B} \]

which evaluates expressions in a given system configuration,

Assuming $I$ to be partial is a way to treat “undefined” during runtime. If $I$ is not defined (for instance because of dangling-reference navigation or division-by-zero), we want to go to a designated “error” system configuration.

- a **transformer** for each action: for each $\text{act} \in \text{Act}$, we assume to have

\[ t_{\text{act}} \subseteq D(C) \times (\Sigma^D_S \times \text{Eth}) \times (\Sigma^D_S \times \text{Eth}) \]

**Transformer**

**Definition.**

Let $\Sigma^D_S$ the set of system configurations over some $\mathcal{S}_0$, $\mathcal{D}_0$, Eth.

We call a relation

\[ t \subseteq D(C) \times (\Sigma^D_S \times \text{Eth}) \times (\Sigma^D_S \times \text{Eth}) \]

a (system configuration) **transformer**.

**Example:**

- $t[u_x](\sigma, \varepsilon) \subseteq \Sigma^D_S \times \text{Eth}$ is
  - the set (!) of the **system configurations**
  - which may result from **object** $u_x$
  - **executing** transformer $t$.
- $t_{\text{skip}}[u_x](\sigma, \varepsilon) = \{(\sigma, \varepsilon)\}$
- $t_{\text{create}}[u_x](\sigma, \varepsilon) : \text{add a previously non-alive object to } \sigma$
**Observations**

- In the following, we assume that each application of a transformer \( t \)
  to some system configuration \((\sigma, \varepsilon)\)
  for object \( u_x \)
  is associated with a set of observations

\[
\text{Obs}_t[u_x](\sigma, \varepsilon) \subseteq 2^{(\mathcal{P}(\mathcal{E}) \cup \{*, +\}) \times \mathcal{S}(\mathcal{E})}.
\]

- An observation \((u_e, u_{dst}) \in \text{Obs}_t[u_x](\sigma, \varepsilon)\)
  represents the information that, as a "side effect" of object \( u_e \)
  executing \( t \) in system configuration \((\sigma, \varepsilon)\),
  the event \( u_e \) has been sent to \( u_{dst} \).

**Special cases:** creation ("*") / destruction ("+”).

---

**A Simple Action Language**

In the following we use

\[
\text{Act}_\mathcal{S} = \{ \text{skip} \}
\]

\[
\cup \{ \text{update}(\text{expr}_1, v, \text{expr}_2) \mid \text{expr}_1, \text{expr}_2 \in \text{Expr}_\mathcal{S}, v \in \text{atr} \}
\]

\[
\cup \{ \text{send}(E(\text{expr}_1, ..., \text{expr}_n), \text{expr}_{dst}) \mid \text{expr}_i, \text{expr}_{dst} \in \text{Expr}_\mathcal{S}, E \in \mathcal{S} \}
\]

\[
\cup \{ \text{create}(C, \text{expr}, v) \mid C \in \mathcal{C}, \text{expr} \in \text{Expr}_\mathcal{S}, v \in V \}
\]

\[
\cup \{ \text{destroy}(\text{expr}) \mid \text{expr} \in \text{Expr}_\mathcal{S} \}
\]

and OCL expressions over \( \mathcal{S} \) (with partial interpretation) as \( \text{Expr}_\mathcal{S} \).
## Transformer Examples: Presentation

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>op</strong></td>
<td><strong>skip</strong></td>
</tr>
<tr>
<td>intuitive semantics</td>
<td><em>do nothing</em></td>
</tr>
<tr>
<td>well-typedness</td>
<td>./.</td>
</tr>
<tr>
<td>semantics</td>
<td>$t_{\text{skip}}[u_x](\sigma, \varepsilon) = {(\sigma, \varepsilon)}$</td>
</tr>
<tr>
<td>observables</td>
<td>$\text{Obs}_{\text{skip}}[u_x](\sigma, \varepsilon) = \emptyset$</td>
</tr>
<tr>
<td>(error) conditions</td>
<td>Not defined if . . .</td>
</tr>
</tbody>
</table>

### Transformer: Skip

<table>
<thead>
<tr>
<th>abstract syntax</th>
<th>concrete syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>op</strong></td>
<td><strong>skip</strong></td>
</tr>
<tr>
<td>intuitive semantics</td>
<td><em>do nothing</em></td>
</tr>
<tr>
<td>well-typedness</td>
<td>./.</td>
</tr>
<tr>
<td>semantics</td>
<td>$t_{\text{skip}}[u_x](\sigma, \varepsilon) = {(\sigma, \varepsilon)}$</td>
</tr>
<tr>
<td>observables</td>
<td>$\text{Obs}_{\text{skip}}[u_x](\sigma, \varepsilon) = \emptyset$</td>
</tr>
<tr>
<td>(error) conditions</td>
<td>Not defined if . . .</td>
</tr>
</tbody>
</table>
Transformer: Update

abstract syntax
update(expr₁, v, expr₂)

concrete syntax
expr₁, v ⇝ expr₂

intuitive semantics
Update attribute v in the object denoted by expr₁ to the value denoted by expr₂.

well-typedness
expr₁ : TC and v : T ∈ atr(C); expr₂ : T;
expr₁, expr₂ obey visibility and navigability

semantics
t_{update}(expr₁,v,expr₂)[u₁x](σ,ε) = \{(σ',ε)\}

where σ' = σ[u → σ(u)[v → I[expr₂](σ, u₂)]] with

\[ u = I[expr₁](σ, u₁) \] (object denoted by expr₁)

observables
Obs_{update}(expr₁,v,expr₂)[u₂] = \{∅\}

(error) conditions
Not defined if I[expr₁](σ, u₂) or I[expr₂](σ, u₂) not defined.

Update Transformer Example

\[ SMC : \]

\[ \begin{array}{c}
\sigma : \\
\begin{array}{l}
x = 4 \\
y = 0 \\
\text{skel} = 0 \\
\end{array}
\end{array} \]

\[ \begin{array}{c}
\sigma' : \\
\begin{array}{l}
x = 5 \\
y = 0 \\
\text{skel} = 0
\end{array}
\end{array} \]

\[ \varepsilon : \]

\[ \begin{array}{l}
u = I[expr₁](σ, u₁) \\
u = σ[x \mapsto \text{skel}] \\
u = \text{skel} + 1
\end{array} \]

\[ = \begin{array}{l}
\text{skel} + 1 \\
\end{array} \]

\[ \begin{array}{l}
\sigma' = \sigma + 1 \\
\epsilon' = \varepsilon + 1
\end{array} \]

\[ \begin{array}{l}
\text{skel} + 1 \\
\text{skel} + 1
\end{array} \]
abstract syntax
\[
\text{send}(E, \ldots, expr_n, expr_{\text{dst}})
\]
concrete syntax
\[
\text{expr}_{\text{dst}} \leftarrow E(\text{expr}_1, \ldots, \text{expr}_n)
\]

intuitive semantics
Object \( u_x : C \) sends event \( E \) to object \( expr_{\text{dst}} \), i.e. create a fresh signal instance, fill in its attributes, and place it in the ether.

well-typedness
\( E \in \mathcal{E} \); \( \text{atr}(E) = \{ v_1 : T_1, \ldots, v_n : T_n \} \); \( \text{expr}_i : T_i, 1 \leq i \leq n \); \( \text{expr}_{\text{dst}} : T_{\text{D}}, C, D \in \mathcal{E} \setminus \mathcal{E} \);
all expressions obey visibility and navigability in \( C \)

semantics
\( (\sigma', \varepsilon') \in t_{\text{send}}(E, \ldots, expr_n, expr_{\text{dst}})[u_x][\sigma, \varepsilon] \)

\( \theta \) if \( \sigma' = \sigma \cup \{ y \mapsto \{ v_i \mapsto d_i | 1 \leq i \leq n \} \}; \varepsilon' = \varepsilon \oplus (u_{\text{dst}}, y) \);
if \( u_{\text{dst}} = I[expr_{\text{dst}}](\sigma, u_x) \in \text{dom}(\sigma) \); \( d_i = I[expr_i](\sigma, u_x) \) for \( 1 \leq i \leq n \);

\( u \in \mathcal{D}(E) \) a fresh identity, i.e. \( u \notin \text{dom}(\sigma) \), and where \( (\sigma', \varepsilon') = (\sigma, \varepsilon) \) if \( u_{\text{dst}} \notin \text{dom}(\sigma) \);

\( \text{observables} \)
\( \text{Obs}_{\text{send}}[u_x] = \{(u, u_{\text{dst}})\} \)

(error) conditions
\( I[expr](\sigma, u_x) \) not defined for any \( expr \in \{ expr_{\text{dst}}, expr_1, \ldots, expr_n \} \)

Send Transformer Example

\( S_{MC} \):

\[
\begin{array}{ccc}
  & /n \! F(x + 1) & \\
  s_1 & \Rightarrow & s_2 \\
\end{array}
\]

\[
\begin{array}{c}
  s_1 \\
  /n \! F(x + 1) \\
\end{array}
\]

\[
\begin{array}{c}
  s_2 \\
  \text{send}[u_x] \\
\end{array}
\]

\[
\begin{array}{c}
  u_1 : C \\
  x = 5 \\
\end{array}
\]

\[
\begin{array}{c}
  u_2 : C \\
  x = 6 \\
\end{array}
\]

\[
\begin{array}{c}
  u_3 : C \\
  x = 7 \\
\end{array}
\]

\[
\begin{array}{c}
  u_{\text{dst}} : C \\
  x = 8 \\
\end{array}
\]

\[
\begin{array}{c}
  \text{send}(E_{\text{dst}}) \\
  \text{send}(E_{\text{src}}) \\
  \text{send}(E_{\text{dst}}) \\
\end{array}
\]

\[
\begin{array}{c}
  \text{FIFO} \\
  \text{FIFO} \\
\end{array}
\]
Sequential Composition of Transformers

- **Sequential composition** $t_1 \circ t_2$ of transformers $t_1$ and $t_2$ is canonically defined as

$$ (t_2 \circ t_1)[u_x](\sigma, \varepsilon) = t_2[u_x](t_1[u_x](\sigma, \varepsilon)) $$

with observation

$$ \text{Obs}_{(t_2 \circ t_1)}[u_x](\sigma, \varepsilon) = \text{Obs}_{t_1}[u_x](\sigma, \varepsilon) \cup \text{Obs}_{t_2}[u_x](t_1(\sigma, \varepsilon)). $$

- **Clear**: not defined if one the two intermediate “micro steps” is not defined.

Transformers And Denotational Semantics

**Observation**: our transformers are in principle the **denotational semantics** of the actions/action sequences. The trivial case, to be precise.

**Note**: with the previous examples, we can capture

- empty statements, skips,
- assignments,
- conditionals (by normalisation and auxiliary variables),
- create/destroy (later),

but not possibly diverging loops.

**Our (Simple) Approach**: if the action language is, e.g. Java, then (syntactically) forbid loops and calls of recursive functions.

**Other Approach**: use full blown denotational semantics.

No show-stopper, because loops in the action annotation can be converted into transition cycles in the state machine.
References

